맵리듀스 퍼포먼스 튜닝하기

2012.11

정재화

GRUTER

About me

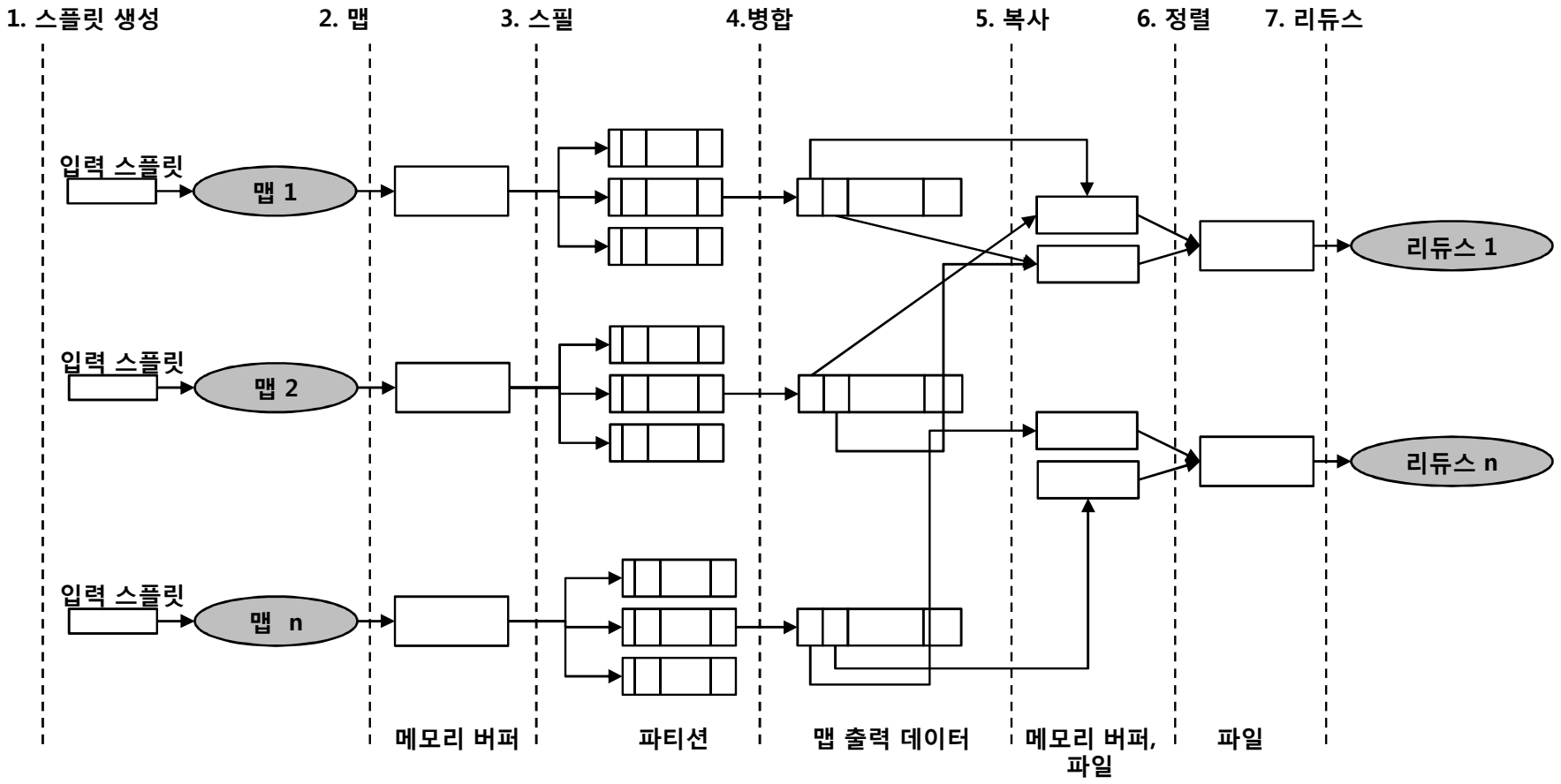
- jhjung@gruter.com
- 현)그루터 (www.gruter.com) 책임 개발자
- 전) 큐릭스, NHN, 엔씨소프트
- www.blrunner.com
- www.twitter.com/blrunner78
- 저서) 시작하세요!하둡 프로그래밍: 기초부터 실무까지 하둡의 모든 것 (위키북스, 2012년10월 출간)

GRUTER

Index

1.1 셔플이란

셔플은 메모리에 저장돼 있는 매퍼의 출력 데이터를 파티셔닝과 정렬을 해서 로컬 디스크에 저장한 후, 네트워크를 통해 리듀서의 입력 데이터로 전달되는 과정을 의미합니다



1.1 셔플이란

속성	내용	기본값
io.sort.mb	맵의 출력 데이터를 저장할 버퍼의 크기	100mb
io.sort.spill.percent	맵의 출력 데이터를 저장하는 버퍼가 일정한 비율에 도달하면 로컬 디스크로 스필을 실행함	0.80 → 80%
io.sort.factor	하나의 정렬된 출력 파일로 병합하기 위한 스필 파일의 개수	10
tasktracker.http.threads	맵의 출력 데이터를 리듀스 태스크에게 제공하기 위한 태스크트래커의 워커 쓰레드 개수. 전체 클러스터에 적용되며, 잡별로 다르게 설정할 수는 없음	40
mapred.reduce.parallel.copies	맵의 출력 데이터를 리듀스 태스크로 복사하기 위한 쓰레드 개수	5

1.1 셔플이란

속성	내용	기본값
mapred.job.shuffle.input.buffer.percent	셔플된 맵의 출력 데이터는 리듀스 태스크 트래커의 메모리 버퍼로 복사됨. 이때 메모리 버퍼로 사용할 리듀스 태스크 트래커의 힙 메모리 사이즈 비율을 설정함	0.70 → 70%
mapred.job.shuffle.merge.percent	리듀스 태스크의 메모리 버퍼에 저장된 맵의 출력 데이터를 스필하기 위한 임계치	0.66 → 80%
mapred.inmem.merge.threshold	리듀스 태스크의 메모리 버퍼에서 처리할 파일 개수가 일정 개수에 도달해도 스필을 실행함.	1000

1.2 셔플 환경 변수 튜닝

- 맵 태스크는 스필 파일의 개수를 줄이는 것이 유리함
- 튜닝 대상 맵리듀스 잡
  - ✓ 입력 데이터 크기: 11G
  - ✓ 작업 유형: Customer CompositeKey 와 GroupKey를 이용한 보조 정렬
  - ✓ 작업 수행 시간: 약13분 소요
- 튜닝 속성
  - ✓ io.sort.mb: 100mb → 200mb
  - ✓ io.sort.factor: 10 → 30
- 맵리듀스 잡 실행
  - ✓ 전체 클러스터의 환경 설정값을 수정하지 않고, 개별적인 잡의 설정값만 변경함
  - ✓ 런타임에서 –D 옵션을 이용해 환경 변수값을 수정함

```
./bin/hadoop jar wikibooks-hadoop-examples.jar
wikibooks.hadoop.chapter06.DelayCountWithDateKey -D io.sort.mb=200 -D
io.sort.factor=30 input delay_count_sort2
```

1.2 셔플 환경 변수 튜닝

- 맵 리듀스 잡 실행 시 Java heap space 오류 발생

```
12/08/27 00:51:45 INFO mapred.JobClient: Task Id :
attempt_201208250324_0040_m_000000_0, Status : FAILED
Error: Java heap space
12/08/27 00:51:45 INFO mapred.JobClient: Task Id :
attempt_201208250324_0040_m_000001_0, Status : FAILED
Error: Java heap space
12/08/27 00:51:47 INFO mapred.JobClient: Task Id :
attempt_201208250324_0040_m_000002_0, Status : FAILED
Error: Java heap space
```

- 태스크 트래커에서 실행하는 JVM의 기본 힙 메모리 크기(mapred.child.java.opts)를 초과했기 때문에 오류가 발생함
  - ✓ mapred.child.java.opts를 300mb 이상으로 설정해서 맵리듀스 잡을 실행해야 함

- mapred.child.java.opts 를 512mb로 수정해서 재실행

```
./bin/hadoop jar wikibooks-hadoop-examples.jar
wikibooks.hadoop.chapter06.DelayCountWithDateKey -D io.sort.mb=200 -D
mapred.child.java.opts=-Xmx512m -D io.sort.factor=30 input delay_count_sort3
```

1.2 셔플 환경 변수 튜닝

- 작업시간: 12분42초 → 9분 50초
- 튜닝 결과 비교

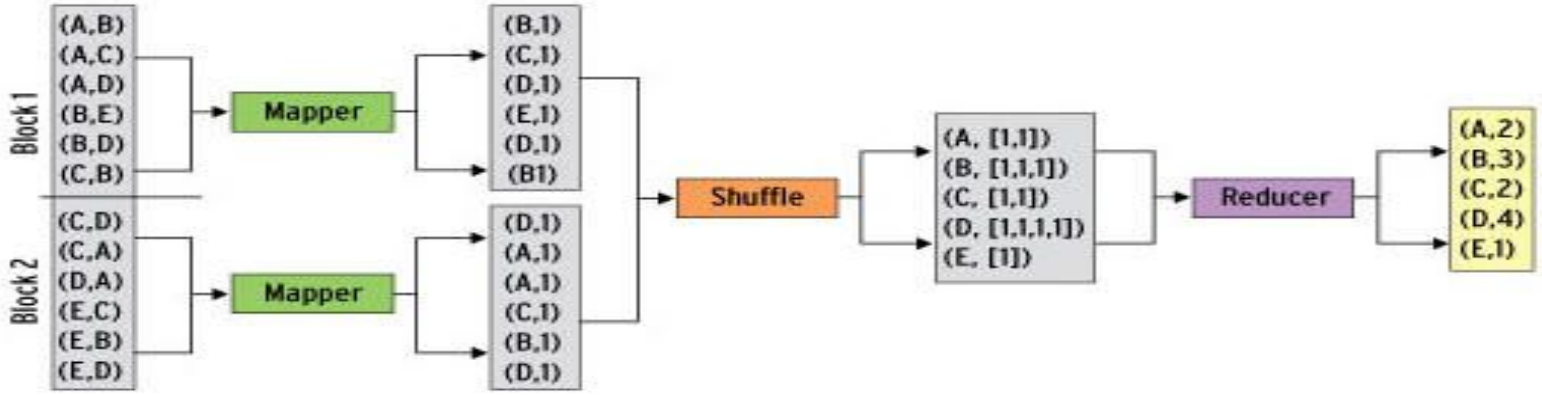
카운터	io.sort.* 적용전	io.sort.* 적용후
SLOTS_MILLIS_MAPS	2,395,970	2136,569
SLOTS_MILLIS_REDUCES	717,843	550,099
FILE_BYTES_READ	5,542,981,602	3,933,998,336
HDFS_BYTES_READ	11,966,801,064	11,966,801,064
FILE_BYTES_WRITTEN	7,690,995,377	6,082,012,298
HDFS_BYTES_WRITTEN	7,269	7,269
Reduce shuffle bytes	2,143,640,736	2,138,251,210
Spilled Records	384,330,903	303,881,809
CPU time spent (ms)	2,097,910	1,721,660
Total committed heap usage (bytes)	38,668,533,760	84,327,792,640
Physical memory (bytes) snapshot	43,280,994,304	82,583,814,144
Virtual memory (bytes) snapshot	105,656,713,216	168,085,508,096

- 셔플 파라미터는 맵리듀스 프로그램 코드 상에서 정렬을 사용하는 경우에 튜닝 효과가 나타나며, 정렬을 사용하지 않는 잡의 셔플 파라미터를 수정하면 기본값을 사용하는 경우보다 성능이 나빠지게 됨

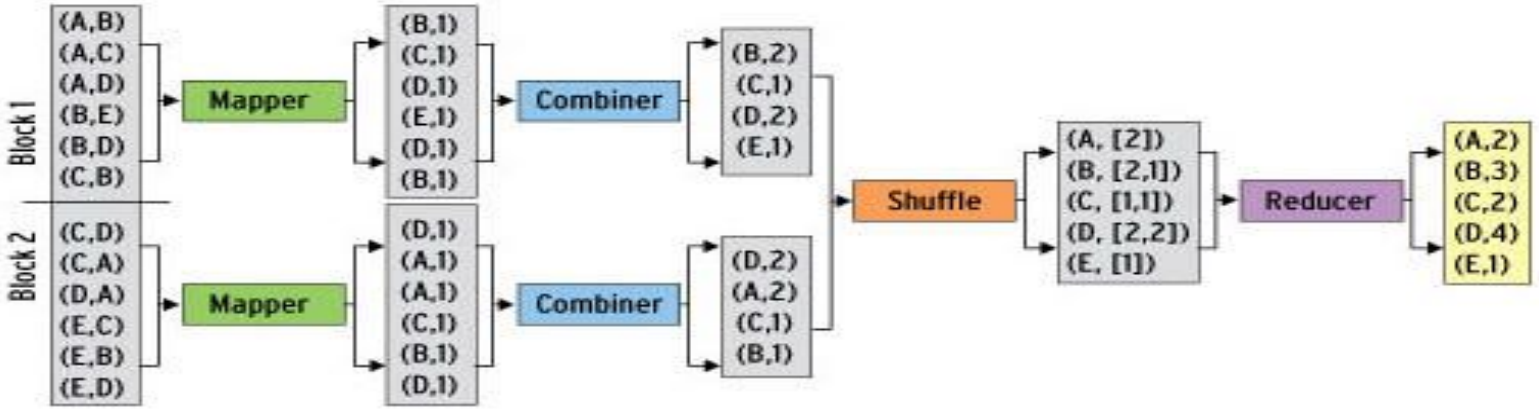
1.3 콤바이너 클래스 적용

- 콤바이너 클래스는 맵의 출력 데이터가 네트워크를 통해 리듀서로 전달되기 전에 매퍼의 출력 데이터의 크기를 줄이는 기능을 수행함

콤바이너 적용전



콤바이너 적용후



✓그림 출처: http://hpc.admin-magazine.com/Articles/MapReduce-and-Hadoop

1.3 콤바이너 클래스 적용

- Job 인터페이스의 setCombinerClass 메소드를 호출해서 적용. 반드시 리듀서 클래스를 파라미터로 설정해야함
  - ✓ ***job.setCombinerClass(DelayCountReducer.class);***
- 튜닝 대상 맵리듀스 잡
  - ✓ 입력 데이터 크기: 11G
  - ✓ 작업 유형: 단순 계산 (wordcount성)
- 튜닝 전보다 약 40초 가량 수행 시간 단축
- 콤바이너 클래스 적용 결과 비교

카운터	콤바이너 적용 전	콤바이너 적용 후
FILE_BYTES_READ	1,453,088,421	16,577
HDFS_BYTES_READ	11,966,801,064	11,966,801,064
FILE_BYTES_WRITTEN	2,219,075,754	411,2660
HDFS_BYTES_WRITTEN	3,635	3,635
Reduce shuffle bytes	761,365,791	8,760
Spilled Records	167,244,625	1,685
CPU time spent (ms)	1,013,810	930,160
Total committed heap usage	3,8565,642,240	38,504,628,224
Reduce input records	57,482,463	578
Physical memory (bytes) snapshot	42,829,991,936	42,876,624,896

1.4 맵 출력 데이터 압축

- 맵 출력 데이터를 압축하면, 파일 I/O와 네트워크 비용이 감소함
- 하둡 클러스터에서 맵 출력 데이터를 압축하기 위해서는, mapred-site.xml에 다음과 같이 정의함

```
<!-- 맵 출력 데이터 압축 사용 여부 -->
 <property>
   <name>mapred.compress.map.output</name>
   <value>true
   </value>
 </property>

 <!-- 클러스터에서 사용할 압축 라이브러리 정의 -->
 <property>
   <name>io.compression.codecs</name>
   <value>
     org.apache.hadoop.io.compress.GzipCodec,
     org.apache.hadoop.io.compress.DefaultCodec,
     org.apache.hadoop.io.compress.BZip2Codec,
     org.apache.hadoop.io.compress.SnappyCodec
   </value>
 </property>
```

1.4 맵 출력 데이터 압축

- 스내피(Snappy)란 구글에서 개발한 압축 라이브러리이며, 하둡 1.02와 0.23 버전부터 정식으로 스내피를 지원함
- 압축률을 최대한 높이기 보다는, 적당한 압축률로 빠르게 압축을 수행하는 것을 목표로 함
- 초당 250MB로 압축을 수행함
- 구글 내부에서는 빅 테이블, 구글 파일 시스템, RPC 시스템 등에 광범위하게 적용됨
- 공식 사이트: http://code.google.com/p/snappy/
- 설치 방법: http://www.blrunner.com/8

1.4 맵 출력 데이터 압축

- 개별적인 맵리듀스 잡에서 맵 출력의 압축은 다음과 같이 코드를 작성함
  - ✓ **Gzip**

```
job.setOutputFormatClass(SequenceFileOutputFormat.class);
SequenceFileOutputFormat.setCompressOutput(job, true);
SequenceFileOutputFormat.setOutputCompressorClass(job, GzipCodec.class);
SequenceFileOutputFormat.setOutputCompressionType(job,
CompressionType.BLOCK);
```

  - ✓ **Snappy**

```
job.setOutputFormatClass(SequenceFileOutputFormat.class);
SequenceFileOutputFormat.setCompressOutput(job, true);
SequenceFileOutputFormat.setOutputCompressorClass(job, SnappyCodec.class);
SequenceFileOutputFormat.setOutputCompressionType(job, CompressionType.BLOCK);
```

1.4 맵 출력 데이터 압축

- 튜닝 대상 맵리듀스 잡
  - ✓ 입력 데이터 크기: 11G
  - ✓ 작업 유형: 단순 계산 (wordcount성)
- Gzip 적용 시 압축을 적용하지 않았을 때보다 작업이 지연되고, Snappy 적용 시 압축을 적용하지 않았을 때보다 작업 시간이 감소함
  - Gzip: 1분 30초 지연
  - Snappy: 10초 단축
- 압축 라이브러리 적용 결과 비교

카운터	압축 미적용	Gzip 적용	Snappy 적용
FILE_BYTES_READ	1,453,088,421	2,838,889	68,863,863
HDFS_BYTES_READ	11,966,801,064	11,966,801,064	11,966,801,064
FILE_BYTES_WRITTEN	2,219,075,754	8,391,610	109,038,190
HDFS_BYTES_WRITTEN	3,635	1,457	2,228
Reduce shuffle bytes	761,365,791	1,485,204	36,082,447
CPU time spent (ms)	1,013,810	1,347,100	990,840
Total committed heap usage	3,8565,642,240	38,536,019,968	38,570,033,152

2. DFS 블록 사이즈 수정

- 맵리듀스 잡은 수행되는 맵 태스크와 리듀스 태스크의 개수에 따라 성능의 영향을 받게 됨
- 같은 사이즈의 파일이라도 더 많은 수의 블록으로 분리되면, 그 만큼 많은 맵 태스크가 수행되어서 빠르게 작업이 수행될 것임
- 예제) 블록 사이즈에 따른 예상 맵 태스크 개수

입력 파일	블록 사이즈(dfs.block.size)	맵 태스크 수
100GB	32MB	(100 * 1024) / 32 = 3200
100GB	64MB	(100 * 1024) / 64 = 1600
100GB	128MB	(100 * 1024) / 128 = 800
100GB	256MB	(100 * 1024) / 256 = 400

2. DFS 블록 사이즈 수정

- 블록 사이즈 수정 방법
  - ✓ Hdfs-site.xml의 dfs-block-size 속성을 수정 (기본값: 64mb)
  - ✓ distcp 코맨드를 이용해 HDFS에 기등록된 파일을 새로운 블록 사이즈의 파일을 복사함

```
./bin/hadoop distcp -Ddfs.block.size=[HDFS 블록 사이즈] [입력 경로] [출력 경로 ]
./bin/hadoop distcp -Ddfs.block.size=$[32*1024*1024] /user/hadoop/input/2008.csv
/user/hadoop/input_32mb/2008.csv
```

- 32MB 블록으로 구성된 파일을 대상으로 잡을 실행할 경우, 64MB 블록으로 구성된 파일로 잡을 실행할 때 보다 2배 이상 작업이 지연됨
  - ✓ 태스크 트래커에서 동시에 실행할 수 있는 맵 태스크 개수는 한정되어 있는데, 이러한 값에 대한 수정 없이 맵 태스크만 과도하게 실행했기 때문임

2. DFS 블록 사이즈 수정

- 태스크 트래커에서 동시에 실행할 수 있는 태스크 개수는 다음과 같음

파라미터	기본값	내용
mapred.tasktracker.map.tasks.maximum	2	하나의 태스크트래커에서 동시에 실행할 수 있는 맵 태스크의 개수
mapred.tasktracker.reduce.tasks.maximum	2	하나의 태스크트래커에서 동시에 실행할 수 있는 리듀스 태스크의 개수

- 맵 태스크가 CPU의 75% 이하를 사용할 경우 최대 맵 태스크 개수를 CPU 개수 보다 약간 초과하게 설정함
  - ✓ CPU 코어가 8개일 경우: 맵 태스크 8개, 리듀스 태스크 2개 혹은 맵 태스크 7개, 리듀스 태스크 3개로 설정함
- 하둡은 적정 리듀스 태스크 개수를 "0.95 * 데이터노드 개수 ~ 1.75 * 데이터노드 개수"로 권장함

2. DFS 블록 사이즈 수정

- CPU 코어 개수 외에 하드 디스크 구성, 입력 데이터 크기, 맵리듀스 잡의 분석 유형 등 다양한 요소를 고려해야 함
- 맵 태스크의 작업이 단순해서 수초 내에 완료되는 경우 블록 사이즈 조정이 효과가 있지만, 수십초 이상 소요될 경우에는 블록 사이즈 조정의 효과가 없음
  - ✓ 태스크 트래커는 자신에게 할당된 태스크를 주기적으로 할당받는 방식임
  - ✓ 태스크 수행시간이 짧은 경우는 할당받는 시간이 오버헤드가 되서 튜닝 효과가 있지만, 태스크 수행시간이 긴 경우에는 할당받는 시간이 잡 전체의 수행시간에 영향을 주지 않음

3. JVM 재사용

- 태스크 트래커는 맵 태스크와 리듀스 태스크를 실행할 때마다 별도의 JVM을 실행함
- 일반적으로 사용자가 인지하지 못할 만큼 빠른 속도로 JVM이 시작됨
- 맵 사이드 조인, 매퍼와 리듀서의 초기화 시간이 지연되는 경우, 짧은 시간안에 많은 매퍼를 생성하는 경우 전체 잡 실행에 영향을 줄 수 있음
- JVM을 재사용할 경우 태스크의 시작 시간을 줄일 수 있음
- mapred.job.reuse.jvm.num.tasks 속성값은 JVM이 실행할 수 있는 최대 태스크 개수임. 기본값은 1로 설정되어 있어, JVM을 재사용하지 않음
- mapred.job.reuse.jvm.num.tasks 속성값을 -1로 설정할 경우 JVM이 실행할 수 있는 태스크 개수에 제한이 없어짐

4. 투기적인 잡 실행

- 투기적인(speculative) 잡 실행이란?

  일정 시간이 지났는데도 계속 실행되는 태스크가 있으면, 해당 태스크가 실행되고 있는 데이터 노드와 다른 데이터 노드에서 태스크를 실행함. 기존 태스크가 먼저 완료되면, 새로 실행한 태스크를 강제 종료하고, 새로 실행한 태스크가 먼저 완료되면, 기존 태스크를 강제 종료함

파라미터	기본값	내용
mapred.map.tasks.speculative.execution	true	잡이 실행되는 중 다른 맵 태스크보다 느리게 동작하는 맵 태스크가 있으면 다른 데이터 노드에서 해당 맵 태스크를 실행합니다.
mapred.reduce.tasks.speculative.execution	true	잡이 실행되는 중 다른 리듀스 태스크보다 느리게 동작하는 리듀스 태스크가 있으면 다른 데이터 노드에서 해당 리듀스 태스크를 실행합니다.

4. 투기적인 잡 실행

- 기본값은 true로 설정되어 있음.
- 잡 인터페이스의 API로 설정 변경이 가능함

메서드	내용
setSpeculativeExecution	job에서 수행하는 모든 task에 대한 투기적인 실행 옵션을 설정합니다.
setMapSpeculativeExecution	job에서 수행하는 map task에 대해 투기적인 실행 옵션을 설정합니다.
setReduceSpeculativeExecution	job에서 수행하는 reduce task에 대해 투기적인 실행 옵션을 설정합니다.

- 설정값을 false로 변경하고, 개별적인 잡에 대해서 true 설정하는 것을 권장함
  - ✓ 태스크의 로직에 문제, HDFS가 아니라 별도 파일 시스템에 read/write를 하는 경우에 투기적인 실행이 하둡 클러스터에 부하를 줄 수 있음

5. DBMS에 데이터 입력하기

- 리듀스 태스크에서 DBMS에 데이터를 입력하는 경우 장애가 발생할 수 있음
  - ✓ 투기적인(Sepcualative) 잡이 true로 설정된 경우
  - ✓ 태스크 fail로 인하여 자동 retry가 발생하는 경우
- 성능과 데이터 정합성 측면에서 리듀스 태스크에서 DBMS에 데이터를 입력하는 것은 지양해야 함
- DBMS에 로딩할 수 있는 파일을 만들어서, DBMS에 로딩하는 것을 추천함